Proactive resource allocation heuristics for robust project scheduling

Kristof Braeckmans, Erik Demeulemeester,
Willy Herroelen and Roel Leus
Research Center for Operations Management,
K.U.Leuven, Belgium
email: <first name>.<family name>@econ.kuleuven.be

October 11, 2005

Abstract

The well-known deterministic resource-constrained project scheduling problem (RCPSP) involves the determination of a *predictive schedule (baseline schedule* or *pre-schedule)* of the project activities that satisfies the finish-start precedence relations and the renewable resource constraints under the objective of minimizing the project duration. This pre-schedule serves as a baseline for the execution of the project. During execution, however, the project can be subject to several types of disruptions that may disturb the baseline schedule. Management must then rely on a *reactive scheduling* procedure for revising or reoptimizing the pre-schedule.

The objective of our research is to develop procedures for allocating resources to the activities of a given baseline schedule in order to maximize its stability. We propose two integer programming based heuristics and report on computational results obtained on a set of benchmark problems.

1 Introduction

The research on resource-constrained project scheduling has significantly expanded over the last few decades. The vast majority of these research efforts focus on the development of exact and heuristic procedures for the generation of a workable *baseline schedule* (*pre-schedule* or *predictive schedule*), assuming complete information and a static and deterministic environment. Such

a baseline schedule is usually constructed by solving the so-called *resource-constrained project scheduling problem* (RCPSP). This problem (problem $m, 1|cpm|C_{max}$ in the notation of Herroelen et al. (2000)) involves the determination of a schedule that satisfies both the zero-lag finish-start precedence constraints between the activities and the renewable resource constraints under the objective of minimizing the project duration (for reviews, we refer to Brucker et al. (1999), Demeulemeester and Herroelen (2002), Herroelen et al. (1998), Kolisch and Hartmann (1999), and Kolisch and Padman (1999)).

A baseline schedule serves a number of important functions, such as facilitating resource allocation, providing a basis for planning external activities (i.e. activities to be performed by subcontractors) and visualizing future work for employees (Aytug et al. (2005), Mehta and Uzsoy (1998)). Pre-schedules are the starting point for communication and coordination with external entities in the company's inbound and outbound supply chain: they are the basis for agreements with suppliers and subcontractors, as well as for commitments to customers.

During execution, however, a project may be subject to considerable uncertainty, which may lead to numerous schedule disruptions. Many types of disruptions have been identified in the literature (we refer to Zhu et al. (2005) and Wang (2005)). Activities can take longer than primarily expected, resource requirements or availability may vary, ready times and due dates may change, new activities may have to be inserted (Artigues and Roubellat (2000)), etc.

When disruptions occur during schedule execution, the baseline schedule needs to be rescheduled. If we wish to explore the aforementioned coordination purposes of a schedule to the best possible extent, it is desirable that the actual start of each activity occurs as closely as possible to its baseline starting time. We refer to *stability* as a quality of the scheduling environment when there is little deviation between the baseline and the executed schedule.

A baseline with express anticipation of disruptions, which is protected against certain undesirable consequences of rescheduling, is called *robust*. The option that we explore in this paper is to introduce stability, also referred to as *solution robustness*, into the baseline schedule through proper allocation of the resources (for more information on solution robust project scheduling, we refer to Herroelen and Leus (2004ab, 2005), Leus and Herroelen (2004)). We develop two integer programming based resource allocation procedures to protect a given baseline schedule against activity duration variability. Schedule stability is measured by the weighted sum of the deviations between the scheduled activity start times in the baseline schedule and the actually realized activity start times during project execution.

The structure of the paper is as follows. Section 2 introduces the ba-

sic definitions and the concept of resource flow networks used to represent the resource allocation decisions. It concludes by a formal statement of the problem under investigation. Section 3 offers a review of the literature. The resource allocation heuristics developed in this paper are described in Section 4. In Section 5 we validate these algorithms against previously developed heuristics on a set of benchmark problems. The last section provides some overall conclusions and suggestions for further research.

2 Resource allocation and resource flow networks

2.1 Basic definitions and notation

We assume a project network consisting of a set N of $n+1$ activities in activity-on-the-node representation with a single zero-duration dummy start node 0 and a single zero-duration dummy end node n. Project activities j ($j = 1, 2, ..., n-1$) have stochastic activity durations $\mathbf{d}_j$, are subject to zero-lag finish-start precedence constraints and require an integer per period amount r_{jk} of one or more renewable resource types k ($k = 1, 2, ..., K$) during their execution. The renewable resource types have a constant per period availability a_k. The dummy activities have zero duration and zero resource usage. We assume a precedence and resource feasible baseline schedule S has been generated using deterministic activity durations d_j. This schedule provides the scheduled activity start times s_j, $j = 0, ..., n$.

Figure 1(a) shows an example project. The number above a node denotes the corresponding activity duration while the number below a node denotes the per period requirement for a single renewable resource type. The resource type has a per period availability of 4 units. Figure 1(b) shows a minimum baseline schedule for the project generated by the branch-and-bound procedure of Demeulemeester and Herroelen (1992, 1997). The corresponding vector of starting times is (0,0,0,2,2,4). This problem instance will be used as an illustrative example throughout the paper.

During project execution, disturbances may occur, causing the actually realized activity start times $\mathbf{s}_j$ to differ from the planned activity start times s_j. It should be attempted to respect the baseline schedule to the best extent possible in order to avoid system nervousness and constant resource rescheduling, in other words, to maintain *stability* in the system. Therefore, we opt for a so-called *railway execution mode* by never starting activities earlier than their prescheduled start time in the baseline schedule. Effectively, the baseline start times become 'release dates' for schedule execution. This

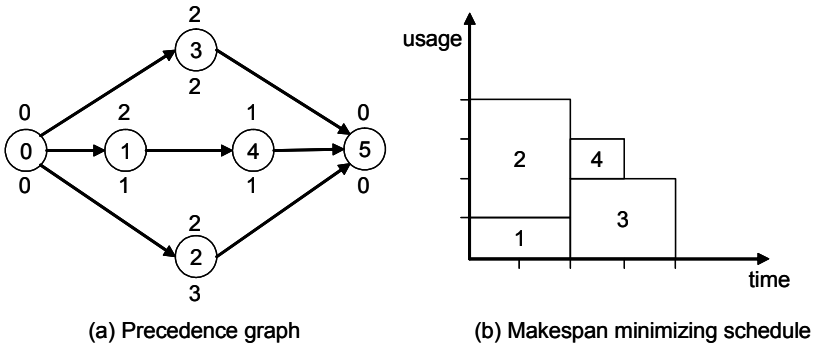

Figure 1: Example project with minimum makespan schedule

type of constraint is inherent to course scheduling, sports timetabling and railway and airline scheduling. In a project setting, activity execution cannot start before the necessary materials have been delivered to the site, and the parties responsible for these prerequisites have normally been communicated as due date the baseline starting time of initial schedule development.

Following Leus (2003), Herroelen and Leus (2004ab) and Leus and Herroelen (2004), we adopt as measure of preschedule stability the *expected weighted deviation in start times* in the actual schedule from those in the baseline schedule. In other words, we aim to minimize $\sum w_j E(\mathbf{s}_j - s_j)$, where E denotes the expectation operator and $w_j \in \mathbb{N}$ denotes the weight of activity j, which is the marginal cost of starting activity j later than planned in the baseline schedule. This may include unforeseen storage costs, extra organizational costs, costs related to agreements with subcontractors or just a cost that expresses the dissatisfaction of employees with schedule changes. We always set $w_0 = 0$; minimization of expected makespan is the special case where $w_j = 0$, $j \neq n$, and $w_n \neq 0$.

2.2 Resource flow networks

The way in which renewable resources are passed on between the various project activities in the baseline schedule can be represented by a *resource flow network* (Artigues and Roubellat (2000), Leus (2003), Leus and Herroelen (2004)). Flow quantity $f_{ijk} \in \mathbb{N}$ is the number of resource units of a certain resource type k, that are transferred from activity i (when it finishes) to activity j (when it starts). We assume that for every resource type k, the sum of all flows *out of* the dummy start activity equals the sum of all flows *into* the dummy end activity, both equal to the total resource availability a_k.

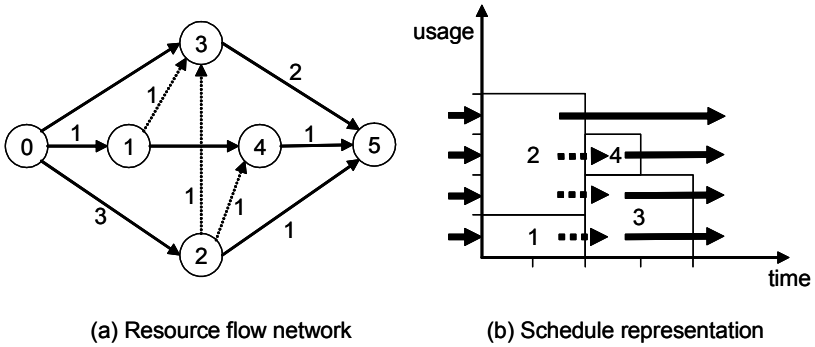

Figure 2: A feasible resource flow network and corresponding schedule

Formally:

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \qquad \forall k \in K \tag{1}$$

Moreover, a feasible resource flow network must satisfy the flow conservation constraints at the intermediate nodes. For every resource type k and for every non-dummy activity $i \neq 0, n$, the sum of flows *into* this activity must equal the sum of flows *out of* this activity, which must be equal to the resource requirement r_{ik}. This results in the following constraint:

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \qquad \forall i \in N \setminus \{0, n\}, \forall k \in K \tag{2}$$

Figure 2(a) shows a possible feasible resource flow network for the example schedule in Figure 1(b). The example project only requires the use of a single resource type, so, in order to simplify notation, we omit the index k. Positive flows f_{ij} are indicated next to each arrow, corresponding to the activity pair (i, j). The non-zero flows are: $f_{01} = 1$; $f_{02} = 3$; $f_{13} = 1$; $f_{23} = 1$; $f_{24} = 1$; $f_{25} = 1$; $f_{35} = 2$; $f_{45} = 1$.

The resource flow network in Figure 2(a) and the schedule representation shown in Figure 2(b) indicate that one of the available resource units is transferred from the end of the dummy start activity to the start of activity 1. The other three available resource units are sent from the end of the dummy start activity to the start of activity 2. At time $t = 2$ (the completion of activities 1 and 2), the resource unit released by activity 1 and one of the resource units released by activity 2 are transferred to the start of activity 3. These resource flows $f_{13} = 1$ and $f_{23} = 1$ impose two extra "resource arcs" indicated by the dotted arcs (1,3) and (2,3), respectively. These arcs induce extra zero-lag finish-start precedence constraints that were not present in

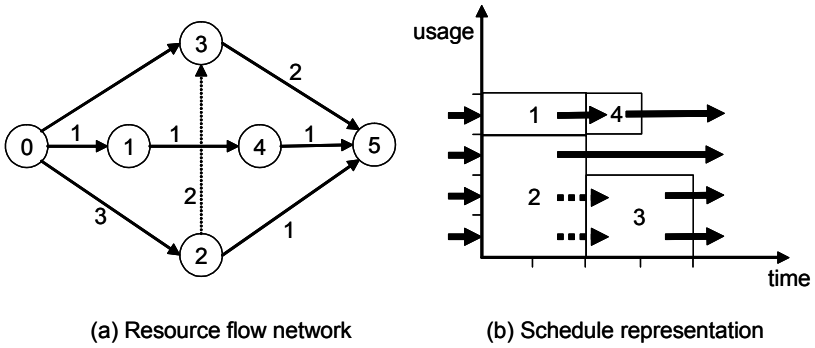

Figure 3: A second feasible resource flow network

the original project network. The second resource unit released by activity 2 is transferred to activity 4 along the extra arc (2,4) which carries the flow $f_{24} = 1$. The third unit is passed on to the dummy end activity 5. Finally, upon their completion, activities 3 and 4 also send their resource units to the dummy end activity 5.

Figure 3 shows an alternative flow network, and as a result an alternative resource allocation, for the same minimal makespan schedule shown in Figure 1(b). The resulting non-zero resource flows are $f_{01} = 1$; $f_{02} = 3$; $f_{14} = 1$; $f_{23} = 2$; $f_{25} = 1$; $f_{35} = 2$; $f_{45} = 1$. Activity 3 now obtains its required resource units from the transfer of two resource units from the completion of activity 2, whereas one resource unit is transferred from the end of activity 1 to the start of activity 4. In this flow network there is only one extra resource arc (2,3), which carries the flow $f_{23} = 2$.

2.3 Activity disruptions and stability

It should be clear that it is often possible to make different resource allocation decisions for the same baseline schedule, each represented by a different resource flow network. The possibility of generating different resource flows for the same baseline schedule may have a serious impact on the robustness of the corresponding reactive scheduling procedure.

In this paper, we assume that uncertainty stems from activity duration variability. When information becomes known about durations $\mathbf{d}_j$ that take on a realization different from d_j, the schedule needs to be repaired. In this schedule repair process, we require the *resource allocation to remain constant*, i.e., the same resource flow is maintained. Such a reactive policy is preferred when specialist resources (e.g. expert staff) cannot be transferred between

activities at short notice, for instance in a multiproject environment, where it is necessary to book key staff or scarce equipment with high set-up cost (e.g. a crane) in advance to guarantee their availability, which makes last-minute changes in resource allocation unachievable (Bowers (1995), Leus and Herroelen (2004)).

Refer again to the resource flow networks shown in Figures 2 and 3. Assume, for example, that project management is uncertain about the duration of activity 2. It is obvious that in this case, the resource flow pattern in Figure 3 is more robust (stable) than the pattern in Figure 2. In Figure 2, a delay in activity 2 immediately affects the scheduled start times of both activities 3 and 4, while for the flow pattern in Figure 3, a delay in activity 2 has no effect on the planned starting time of activity 4.

2.4 Formal problem statement

Given a certain baseline schedule S with activity start times $s_0, ..., s_n$, our objective is to generate the resource flows f_{ijk} such that the stability of the baseline schedule is maximized. Formally:

[Problem $P1$]

$$\text{minimize} \sum_{j \in N} w_j E(\mathbf{s}_j - s_j) \tag{3}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \qquad \forall k \in K \tag{4}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \qquad \forall i \in N \setminus \{0, n\}, \forall k \in K \tag{5}$$

$$\mathbf{s}_j = max(s_j, max_{i \in Pred_j}(\mathbf{s}_i + \mathbf{d}_i)), \qquad \forall j \in N \tag{6}$$

$$f_{ijk} \in \mathbb{N}, \qquad \forall i, j \in N; \forall k \in K \tag{7}$$

The objective function in Eq. (3) is to maximize schedule stability, i.e., to minimize the weighted expected deviation between planned and realized activity start times. Eqs. (4)-(5), shown earlier as Eqs. (1)-(2), are the flow feasibility constraints imposed on a feasible resource flow network . Eqs. (6) specify the railway scheduling reactive policy: $\mathbf{s}_j$, the realized start time of activity j, should be the maximum of the planned start time s_j in the baseline schedule and the maximum finish time of the predecessors $Pred_j$ of activity j in the resource flow network. Eqs. (7) impose integrality on the flow variables.

Problem $P1$ has been shown to be ordinarily NP-hard by Leus (2003) for the single disruption case (for additional NP-hardness proofs of a number of machine scheduling problems with stability objective, we refer to Leus and Herroelen (2005)).

3 Algorithms for stable resource allocation

3.1 Literature overview

3.1.1 Generating feasible resource flows

Artigues and Roubellat (2000) present a simple method to generate a feasible resource flow by extending a parallel schedule generation scheme to derive the flows during scheduling. The algorithm iteratively reroutes flow quantities until a feasible overall flow is obtained. The allocation routine can easily be uncoupled from the schedule generation. For all resource types k, flow f_{0nk} is initialized with value a_k, all other flows are set to 0. We define δ as the set of time instants in the input schedule that correspond with activity start or finish times: $\delta : t \in \delta$ if $\exists j \in N : t = s_j$ or $t = e_j = s_j + d_j$.

The algorithm then runs as follows:

for increasing i in δ **do**
  for $j := 1$ to $(n-1)$ **do**
    if $(s_j == i)$
      for every resource type k **do**
        $req_k = r_{jk}$;
        $m := 0$;
        while $(req_k > 0)$ **do**
          if $e_m \leq s_j$
            q:=min$(req_k, flow_{mnk})$;
            $req_k - = q$;
            $flow_{mnk} - = q$;
            $flow_{mjk} + = q$;
            $flow_{jnk} + = q$;
          $m + +$;

This algorithm just tries to generate a feasible resource flow network and does not aim at maximizing schedule stability or any other measure of performance. It will be used as the worst-case benchmark in the computational experiment described in Section 4.

3.1.2 Branch-and-bound

Leus (2003) and Leus and Herroelen (2004) propose a branch-and-bound model for resource allocation for projects with variable activity durations. The allocation is required to be compatible with a deterministic baseline schedule and the objective is the stability objective given by Eq. (3). Constraint propagation is applied during the search to accelerate the algorithm. The authors obtain computational results on a set of randomly generated networks. However, they restrict their attention to a single resource type and assume exponential activity disruption lengths. Extension to multiple resource types would require a revision of the branching decisions taken by the branch-and-bound procedure and the consistency tests involved in the constraint propagation.

3.1.3 Chained form partial order schedules

Policella (2005) (see also Policella et al. (2004ab)) introduces a two-stage approach, where in a first step, a precedence feasible early start schedule is generated, and then, in the second stage, a procedure referred to as *chaining* is applied to transform this early start schedule into a *chained Partial Order Schedule* (POS). They define a *Partial Order Schedule* (*POS*) as a set of solutions for the RCPSP that can be compactly represented by a temporal graph $G(N, A \cup A_R)$, which is an extension of the precedence graph $G(N, A)$, where N denotes the set of nodes (activities) and A denotes the precedence arcs, with a set of additional arcs A_R, introduced to remove the so-called minimal forbidden sets. A *minimal forbidden set* (Igelmund and Radermacher (1983ab)) is defined for a RCPSP instance as the minimal set of precedence unrelated activities which cannot be scheduled together due to the resource constraints. The set {2,3} is a minimal forbidden set for the RCPSP instance shown in Figure 1(a). Adding the extra arc (2,3) would be sufficient to remove the minimal forbidden set and the resulting *earliest start schedule* of the *POS* in which the activities are started as early as possible would be both time and resource feasible. Apparently, the makespan minimizing schedule shown in Figure 1(b) corresponds to such a time and resource feasible earliest start schedule. The chaining procedure for the generation of a *chained POS* runs as follows:

Sort all activities according to their start times in the input schedule
Initialize all chains empty
for each resource type k **do**
  for each activity j **do**
    for 1 to r_{jk} **do**

$m \leftarrow SelectChain(j,k)$;
$last(m) \leftarrow$ last activity in chain m;
add constraint $last(m) \prec j$;
$last(m) \leftarrow j$;

return *chained POS*.

The first step sorts all activities in increasing order of their starting times in the precedence feasible early start schedule. Then the activities are incrementally allocated on the different chains. Where an activity requires more than one unit of one or more resource types, it will be allocated to a number of chains equal to the overall number of resource units it needs.

The function $SelectChain(j,k)$ is the core of the procedure. In its *Basic Chaining* form, it chooses for each activity the first available chain of its required resource type k (given an activity j, a chain m is available if the end time of the last activity allocated on it, $last(m)$, is not greater than the start time of activity j). Assuming that the schedule of Figure 1(b) is taken as input, the procedure takes activity 1 as first activity on the list and randomly selects for its unit resource requirement, for example, the first available chain <0,1>. Activity 1 is then the last activity on this first chain. The second activity on the list, activity 2, requires the remaining three resource units. Activity 2 is the last activity on each of the three chains <0,2>. Activity 3 requires two units of the resource. Assume that the procedure randomly selects the chain <0,1> to transfer the first resource unit and the second chain <0,2> to pass on the second resource unit. Activity 4 requires one resource unit. Assume that the procedure randomly selects the third chain. In this way, the *Basic Chaining* procedure yields the *chained POS* shown in Figure 4.

The resource allocation shown by the *chained POS* in Figure 4 happens to be identical to the resource allocation represented by the resource flow network of Figure 2. Resource flow networks and chained *POS*s are related concepts: a resource flow network is determined globally for all resource types k, whereas Policella's chains are separately computed for every resource type k.

Note that activities 1 and 4 are situated on different chains. Their mutual precedence constraint ties together the execution of the first and the third chain and as such destroys their independency. Such interdependencies, or *synchronization points*, tend to degrade the stability of the schedule. Although there is no resource dependency between activity 1 situated on chain 1 and activity 4 situated on chain 3, the execution of both chains is not independent. A disturbance in activity 1 on chain 1 will affect chain 3. In order to reduce the number of such synchronization points, Policella et al. develop two additional heuristics ISH and ISH^2.

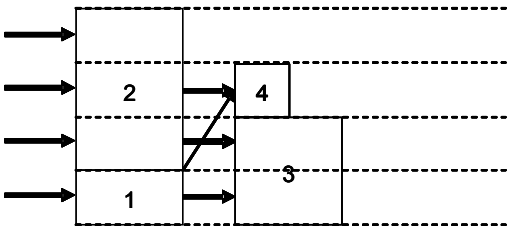

Figure 4: *Chained POS*

ISH tries to favor the allocation of activities to common chains by allocating an activity j according to the following four steps:

1. an initial chain m is randomly selected from among those available for activity j and the constraint $last(m) \prec j$ is imposed;

2. if activity j requires more than one resource unit, then the remaining set of available chains is split into two subsets: the set of chains which has $last(m)$ as last element, $C_{last(m)}$, and the set of chains which does not, $C^{-}_{last(m)}$;

3. to satisfy all remaining resource requirements, activity j is allocated first to chains belonging to the first subset, $m' \in C_{last(m)}$, and,

4. in case this set is not sufficient, the remaining units of activity j are then randomly allocated to the first available chains, m'', of the second subset, $m'' \in C^{-}_{last(m)}$.

Assume that ISH in making the allocation decision for activity 3 in the problem instance of Figure 1 randomly selects the second chain <0,2> imposing the constraint $2 \prec 3$. As activity 3 requires more than one resource unit, the set of available chains is split into two subsets C_2 and C_2^{-}, and activity 3 is allocated to the first available chain in C_2, i.e., chain 3. If activity 4 is then randomly allocated to the third available chain <0,2>, the chained *POS* of Figure 5 is generated. As can be seen, activities 2 and 3 share the second and the third chain.

ISH^2 tries to minimize the interdependencies by replacing the first step of ISH with a more informed choice that takes into account existing ordering relations with those activities already allocated in the chaining process. More precisely, step 1 of ISH is replaced by the following sequence of steps:

1. the chains m for which their last element $last(m)$ is already ordered with respect to activity j are collected in the set P_j;

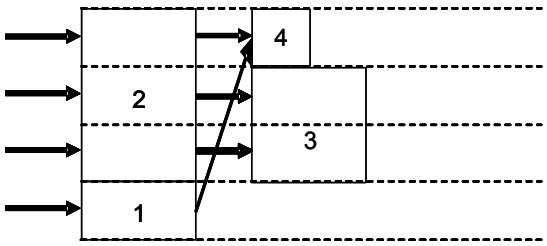

Figure 5: Chained *POS* with common chains

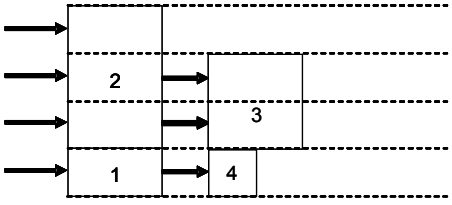

Figure 6: Chained *POS* with removed synchronization point

2. if $P_j \neq \varnothing$ a chain $m \in P_j$ is randomly picked, otherwise a chain m is randomly selected among the available ones;

3. a constraint $last(m) \prec j$ is imposed;

4. continue with steps 2, 3 and 4 of procedure ISH.

Application of ISH^2 on the problem instance of Figure 1 may proceed as follows. Assume activity 1 is randomly allocated to the first chain . Similarly, assume that activity 2 is allocated to the three remaining available chains. The *POS* has imposed the constraint $2 \prec 3$ in order to remove the minimal forbidden set, so that P_3 contains chains 2, 3 and 4. Assume that chain 2 is selected. Continuing with step 2 and the remaining steps of ISH, the remaining resource unit for activity 3 will be taken from chain 3. The algorithm finds that chain 1 already has its last element activity 1 ordered with activity 4, hence P_4 contains chain 1 and activity 4 is situated on chain 1. This yields Figure 6, where there is no longer a synchronization point.

Policella et al. measure schedule robustness using two metrics, fluidity and flexibility. The *fluidity* metric is taken from Cesta et al. (1998) and defined as follows:

$$fldt = \sum_{q \neq r} \frac{Slack(q,r)}{H \times n \times (n-1)} \times 100 \tag{8}$$

where H is the horizon of the problem, n is the number of activities and $Slack(q, r)$ is the width of the allowed distance interval between the end time of activity q and the start time of activity r. This metric characterizes the fluidity of a solution, i.e., the ability to absorb temporal variation in the execution of activities. The hope is that the higher the value of $fldt$, the less the risk of a domino effect, and the higher the probability of localized changes.

The second metric is taken from Aloulou and Portmann (2003) and is called *flexibility*, $flex$. This measure counts the number of pairs of activities in the solution that are related by simple precedence constraints. The rationale for this measure is that when two activities are not related it is possible to move one without moving the other one. The higher the value of $flex$ the lower the degree of interaction among the activities.

Policella et al. do not directly optimize for $fldt$ and $flex$. They apply an iterative sampling search in which they execute the chaining operator described above a number of times from the same initial schedule and pick the best solution with respect to $fldt$ and $flex$.

3.2 IP-based algorithms

As was mentioned above, Problem $P1$ is an NP-hard problem. In this section we describe two heuristic algorithms based on alternative linear integer programming formulations that aim at avoiding the use of stochastic variables.

3.2.1 Minimize the number of extra arcs

When we compare the two suggested solutions for our example problem instance shown in Figures 2 and 3, we see that the more stable solution of Figure 3 imposes fewer extra precedence relations (the dashed resource arcs in the figure). The mixed integer programming model presented in this section aims at minimizing the number of extra arcs imposed by the resource allocation decisions. We define a binary integer variable x_{ij}, taking the value 1 if there is a precedence relationship between activities i and j, 0 otherwise. Minimizing the sum of these x_{ij} variables then boils down to minimizing the total number of additional precedence relations. This results in problem $MinEA$:

[Problem $MinEA$]

$$\text{minimize} \sum_{i \in N} \sum_{j \in N} x_{ij} \tag{9}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \qquad \forall k \in K \tag{10}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \qquad \forall i \in N \setminus \{0, n\}, \forall k \in K \tag{11}$$

$$f_{ijk} \leq M x_{ij}, \qquad (i,j) \in PEA, \forall k \in K \tag{12}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i,j \in N \tag{13}$$

$$f_{ijk} \in \mathbb{N}, \qquad \forall i,j \in N; \forall k \in K \tag{14}$$

The objective function (9) minimizes the number of extra arcs imposed by the resource allocation decisions. Constraints (10) and (11) are again the flow feasibility constraints shown earlier as Eqs. (1)-(2). Eqs. (12), with M a sufficiently large integer, impose extra arcs linking nodes i and j when needed. As soon as, for any resource type k, a resource flow f_{ijk} takes a value strictly larger than zero, the corresponding x_{ij} variable is set equal to 1. This constraint is defined for every activity pair (i,j) in the set of *possible extra arcs* (PEA). This set consists of all pairs of activities (i,j), except those pairs that are already directly or indirectly precedence related in the initial project network, or the pairs that can never be precedence related, due to their starting times in the baseline schedule. In our example instance of Figure 1, $PEA = \{(1,3),(2,3),(2,4)\}$ as activities 1 and 4 are already precedence related in the project network and activities 1 and 2, respectively 3 and 4 share the same start times in the baseline schedule. Eqs. (13) define the 0-1 decision variables, while Eqs. (14) impose integrality conditions on the flow variables.

3.2.2 Maximize the total float

For every activity $j \in N$, we define the scheduled total float TF_j as the total amount of time by which a planned activity starting time s_j in the baseline schedule can be delayed without violating the project due date ω. Formally, the definition looks as follows:

$$TF_n = \omega - s_n \tag{15}$$

$$TF_j = s_n + TF_n - (s_j + max_{paths\ P:j \to n}(\sum_{p \in P} d_p)), \qquad \forall j \in N \setminus n \tag{16}$$

where ω is the project due date and d_p is the deterministic duration of activity p.

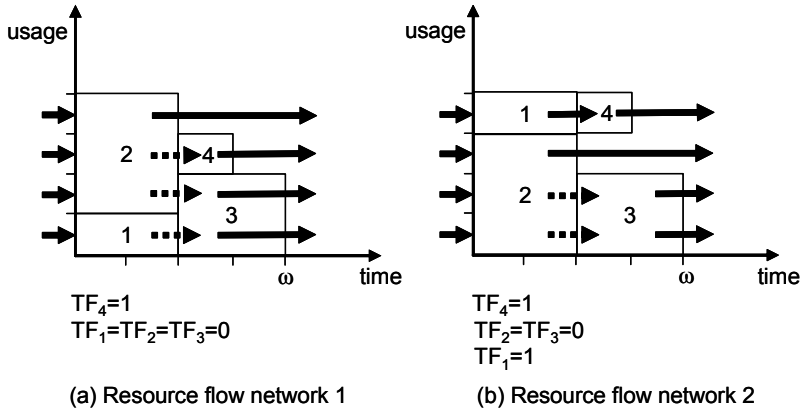

Figure 7: Comparing the total float

It is expected that maximizing the total float over all activities, would increase the stability in the resource flow network. In Figure 7, we compare the total float values for the two resource flow networks of Figures 2 and 3. The sum of the total float values for the schedule shown in Figure 7(b), the more stable schedule, equals 2, whereas for the schedule in Figure 7(a), the total float values only sum up to 1.

We can now formulate Problem $MaxTF$ as follows:

[Problem $MaxTF$]

$$\text{maximize} \sum_{j \in N} w_j TF_j \tag{17}$$

subject to

$$\sum_{j \in N} f_{0jk} = \sum_{j \in N} f_{jnk} = a_k, \qquad \forall k \in K \tag{18}$$

$$\sum_{j \in N} f_{ijk} = \sum_{j \in N} f_{jik} = r_{ik}, \qquad \forall i \in N \setminus \{0, n\}, \forall k \in K \tag{19}$$

$$f_{ijk} \leqslant M x_{ij}, \qquad \forall (i,j) \notin TA, \forall k \in K \tag{20}$$

$$s_i + d_i + TF_i \leqslant s_j + TF_j, \qquad \forall (i,j) \in A \tag{21}$$

$$s_i + d_i + TF_i \leqslant s_j + TF_j + M(1 - x_{ij}), \qquad \forall (i,j) \notin TA \tag{22}$$

$$TF_n = 0 \tag{23}$$

$$x_{ij} \in \{0,1\}, \qquad \forall i,j \in N \tag{24}$$

$$f_{ijk} \in \mathbb{N}, \qquad \forall i,j \in N; \forall k \in K \tag{25}$$

The objective function (17) is to maximize the sum of the weighted scheduled total floats. Eqs. (18)-(19) are again the flow feasibility constraints that define the resource flow network. Eqs. (20) and (22) are defined for all activity pairs that are not in the set of transitive arcs TA. These are all activity pairs that are not precedence related. The variable x_{ij} in Eqs. (20), where M is a sufficiently large number and TA is the set of transitive arcs, is set to 1 as soon as the corresponding resource flow variable f_{ijk} takes a value strictly larger than 0. When this happens, Eqs. (22) is binding, and becomes identical to constraints (21) that are imposed on all precedence related activity pairs in the project network.

Eqs. (21) need some additional clarification. By setting a resource flow $f_{ijk} > 0$, a precedence relation is imposed between activities i and j. Now, assume that the $f_{ijk} > 0$ are chosen such that there exists a path P': $i_1 \rightarrow i_2 \rightarrow ... \rightarrow i_k \rightarrow n$. Applying Eqs. (21) to the activities on this path yields:

$$\begin{aligned}
s_{i1} + d_{i1} + TF_{i1} &\leqslant s_{i2} + TF_{i2} \\
s_{i1} + d_{i1} + d_{i2} + TF_{i1} &\leqslant s_{i2} + d_{i2} + TF_{i2} \\
s_{i1} + d_{i1} + d_{i2} + TF_{i1} &\leqslant s_{i3} + TF_{i3} \\
&\vdots \\
s_{i1} + \sum_{i \in P': i_1 \rightarrow n} d_i + TF_{i1} &\leqslant s_n + TF_n \\
\text{which can be rewritten as } TF_{i1} &\leqslant s_n + TF_n - (s_{i1} + \sum_{i \in P': i_1 \rightarrow n} d_i).
\end{aligned}$$

The last inequality corresponds to the total float as defined by Eq. (16), on the condition that we maximize the total float variables in the objective function. The longest path from $i_1 \rightarrow n$ determines TF_{i_1} if we maximize TF_{i_1}.

Eq. (23) defines the total float of the dummy end node TF_n to be zero. In this case, we assume that the starting time s_n of the zero-duration dummy end activity equals the project deadline ω. Eqs. (24) define the 0-1 variables, while Eqs. (25) impose integrality conditions on the resource flows.

4 Computational results

All computational results have been obtained on a personal computer equipped with a Pentium IV, 2.4 GHZ processor. The algorithm by Artigues and Roubellat (2000) described in Section 3.1.1, and the three algorithms developed by Policella et al., i.e., *Basic Chaining*, ISH and ISH^2,described above

in Section 3.1.3, have been coded in C++. Problems $MinEA$ and $MaxTF$ are solved using the callable libraries of ILOG's CPLEX 8.0.

The weights w_j for each non-dummy activity $j \in \{1, 2....n-1\}$ are drawn from a discrete triangular distribution with $P(w_j = q) = (21 - 2q)\%$ for $q \in \{1, 2....10\}$. This distribution results in a higher occurrence probability for low weights and in an average weight $w_{avg} = 3.85$. The weight w_n of the dummy end activity denotes the marginal cost of violating the project due date and is fixed at $\lfloor 10 \times w_{avg} \rfloor = 38$. For an extensive evaluation of the impact of the activity weights, we refer to Van de Vonder et al. (2004, 2005).

For each activity, the realized activity duration is drawn from a right-skewed beta-distribution with parameters 2 and 5 and an expected value equal to the deterministic activity duration. The minimum and maximum values of this distribution equal 0.5 times and 2.25 times the expected activity duration, respectively.

All procedures have been tested on the 480 networks of the J30 instance set of PSPLIB (Kolisch and Sprecher (1997)). The baseline schedule is generated by the makespan minimizing branch-and-bound algorithm of Demeulemeester and Herroelen (1992, 1997). For each instance and procedure, 100 simulation runs have been made.

	Stability	Stability ($w_j = 1, \forall j \in N$)	CPU time (s)
Artigues	175.16	29.06	0.0646×10^{-3}
Basic Chaining	115.16	17.54	0.133×10^{-3}
ISH_{flex}	115.41	17.62	0.0313×10^{-3}
ISH^2_{flex}	120.93	18.51	0.0625×10^{-3}
$MinEA$	154.55	24.07	2.296
$MaxTF$	107.69	15.79	0.116

Table 1: Computational results

The results of the computational experiment are shown in Table 1. The second column with header *Stability* lists the average stability cost ($\sum w_j E(\mathbf{s}_j - s_j)$) obtained for each heuristic resource allocation procedure over the 100 simulation runs for each of the 480 J30-problem instances of the PSPLIB. Because neither Artigues and Roubellat (2000) nor Policella et al. (2004) take into account the activity weights w_j in making the resource allocation decisions, we also show in the third column the average stability cost results obtained with all activity weights w_j set to 1, i.e., $\sum E(\mathbf{s}_j - s_j)$.

The procedure by Artigues and Roubellat (2000) shows the worst performance for both stability measures. This is according to expectations, because this polynomial time algorithm which was used as worst-case benchmark,

only aims at producing a random feasible resource flow network, without any stability objective.

The *MaxTF* heuristic scores best on both stability measures. *MinEA* has the second worse performance. This does not come as a surprise. The *MinEA* model just minimizes the number of additional precedence relations, without taking into account their impact on the stability cost of the schedule. It is possible that the addition of a single extra direct precedence relationship between two activities induces several indirect precedence relationships, and by doing so, favors the propagation of disruptions throughout the schedule.

The second best performance on both stability measures is obtained by *Basic Chaining*. The performance of this heuristic is comparable to ISH and better than ISH^2. At first sight, this may seem surprising since Policella et al. found ISH and ISH^2, developed in order to maximize the use of combined chains and to reduce the number of synchronization points, to outperform *Basic Chaining* on their fluidity and flexibility metrics. A possible explanation for this performance behaviour can be found in the fact that the Policella heuristics make the allocation decisions seperately for each individual resource type. When dealing with RCPSP instances requiring several renewable resource types, as is the case in our computational experiment, the algorithms may create totally different resource allocations for the different resource types. The *Basic Chaining* procedure was found to generate more similar resource allocations for the different resource types than ISH and ISH^2.

As for the computational requirements, *MinEA* and *MaxTF* are outperformed by the other heuristics, although the average CPU time is still very small, only a fraction of a second for *MaxTF*. This is due to the fact that CPLEX, called to solve each problem instance to optimality, needs more CPU time than the other polynomial time heuristics.

5 Conclusions and suggestions for further research

In this paper, we have offered a formal description of the resource allocation problem under the stability objective of minimizing the sum of the weighted deviations between the planned activity start times in the baseline schedule and the actually realized activity start times during project execution. Our review of the literature revealed that research efforts in this area are still in a burn-in phase.

We have presented two new heuristics based on surrogate MIP formula-

tions of the basic strongly NP-hard problem. The *MinEA* heuristic minimizes the extra precedence relations imposed by the resource allocation decisions, whereas the *MaxTF* heuristic maximizes the sum of the total schedule floats over all activities.

The performance of *MinEA* and *MinTF* has been evaluated against three previously developed heuristics on a set of randomly generated benchmark problems. *MaxTF* obtained the best performance on the stability objective with an average computational requirement of only a fraction of a second on a 2.4 GHz processor. For well-defined reasons, the rather greedy and myopic *MinEA* heuristic ranks rather poor. The runner up is the *Basic Chaining* procedure developed by Policella et al., yielding the best performance among the Policella heuristics.

The resource allocation procedures have all been tested using an input baseline schedule generated by the makespan minimizing brand-and-bound procedure of Demeulemeester and Herroelen (1992, 1997). A minimum makespan schedule is a tight schedule, that is less protected against disruptions in the activity durations and hence, not really stable with regards to our stability measure. A future research track is to rely on other, more robust baseline schedules in order to obtain computational results for larger problem instances and to gain additional insight into the mutual interdependence of the structure of the input baseline schedule and the resource allocation procedure used.

Future research is planned on the development of new exact and heuristic procedures for stable resource allocation. As an example of the latter, and as an alternative for the *maxTF* heuristic, a model could be developed that aims at maximizing the total sum of the scheduled free floats, $\sum_{j \in N} FF_j$. The scheduled free float of an activity j, FF_j, is defined as the allowable delay in the activity finish time without affecting the possible start time of its immediate successors in the schedule. We expect such a model to perform very well on the stability measure $\sum w_j E(\mathbf{s}_j - s_j)$ used in this paper.

Acknowledgements

This research has been supported by Project OT/03/14 of the Research Fund K.U.Leuven.

6 References

Aloulou, M.A. and Portmann, M.C. (2003). An efficient proactive reactive scheduling approach to hedge against shop floor disturbances. *Proceedings of the 1st Multidisciplinary Conference on Scheduling: Theory and Applications* (MISTA 2003), 337-362.

Artigues, C. and Roubellat, F. (2000). A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple modes. *European Journal of Operational Research*, 127, 294-316.

Aytug, H., Lawley, M., McKay, K. Mohan, S. and Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161, 86-110.

Bowers, J.A. (1995). Criticality in resource constrained networks. *Journal of the Operational Research Society*, 46, 80-91.

Brucker, P., Drexl, A., Möhring, R., Neumann, K. and Pesch, E. (1999). Resource-constrained project scheduling: Notation, classification, models and methods. *European Journal of Operational Research*, 112, 3-41.

Cesta, A., Oddi, A. and Smith, S.F. (1998). Profile based algorithms to solve multiple capacitated metric scheduling problems. *Proceedings of the 4th International Conference on Artificial Intelligence Planning Systems*, AFIPS-98, 214-223.

Demeulemeester, E. and Herroelen, W. (1992). A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management Science*, 38, 1803-1818.

Demeulemeester, E. and Herroelen, W. (1997). New benchmark results for the resource-constrained project scheduling problem. *Management Science*, 43, 1485-1492.

Demeulemeester, E. and Herroelen, W. (2002). *Project scheduling - A research handbook,* International Series in Operations Research & Management Science, Vol. 49, Kluwer Academic Publishers, Boston.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (1998). Resource-constrained scheduling: A survey of recent developments. *Computers and Operations Research*, 25, 279-302.

Herroelen, W., De Reyck, B. and Demeulemeester, E. (2000). On the paper "Resource-constrained project scheduling: Notation, classification, models and methods" by Brucker et al.. *European Journal of Operational Research*, 128, 221-230.

Herroelen, W. and Leus, R. (2004a). The construction of stable project baseline schedules. *European Journal of Operational Research*, 156, 550-565.

Herroelen, W. and Leus, R. (2004b). Robust and reactive project scheduling: A review and classification of procedures. *International Journal of Production Research*, 42, 1599-1620.

Herroelen, W. and Leus, R. (2005). Project scheduling under uncertainty: Survey and research potentials, *European Journal of Operational Research*, 165, 289-306.

Igelmund, G. and Radermacher, F.J. (1983a). Preselective strategies for the optimization of stochastic project networks under resource constraints. *Networks*, 13, 1-28.

Igelmund, G. and Radermacher, F.J. (1983b). Algorithmic approaches to preselective strategies for stochastic scheduling problems. *Networks*, 13, 29-48.

Kolisch, R. and Hartmann, S. (1999). Heuristic algorithms for solving the resource-constrained project scheduling problem: Classification and computational analysis. In Weglarz, J. (Ed.). *Project scheduling: Recent models, algorithms and applications*, Kluwer Academic Publishers, Boston.

Kolisch, R. and Padman, R. (1999). An integrated survey of deterministic project scheduling. *Omega*, 49, 249-272.

Kolisch, R. and Sprecher, A. (1997). PSPLIB - A project scheduling library. *European Journal of Operational Research*, 96, 205-216.

Leus, R. (2003). *The generation of stable project plans - Complexity and exact algorithms*, Ph.D. thesis, K.U.Leuven, Belgium.

Leus, R and Herroelen, W. (2004). Stability and resource allocation in project planning. *IIE Transactions*, 36, 667-682.

Leus, R. and Herroelen, W. (2005). The complexity of machine scheduling for stability with a single disrupted job. *Operations Research Letters*, 33, 151-156.

Mehta, S. and Uzsoy, R. (1998). Predictive scheduling of a job shop subject to breakdowns. *IEEE Transactions on Robotics and Automation*, 14, 365-378.

Policella, N. (2005). *Scheduling with uncertainty - A proactive approach using partial order schedules*, unpublished Ph.D. thesis, Università degli Studi di Roma "La Sapienza", Italy.

Policella, N., Oddi, A., Smith, S.F. and Cesta, A. (2004a). Generating robust partial order schedules. In Wallace, M. (Ed.) (2004). Principles and practice of constraint programming. *Lecture Notes in Computer Science*, 3258, 496-511.

Policella, N., Smith, S.F., Cesta, A. and Oddi, A. (2004b). Generating robust schedules through temporal flexibility. *Proceedings of the 14th International Conference on Automated Planning & Scheduling*, ICAPS'04, AAAI, 209-218.

Van de Vonder, S., Demeulemeester, E. and Herroelen, W. (2004). An investigation of efficient and effective predictive-reactive project scheduling procedures. Research Report 0466, Department of Applied Economics, K.U.Leuven, Belgium.

Van de Vonder, S., Demeulemeester E., Herroelen, W. and Leus, R. (2005). The trade-off between stability and makespan in resource-constrained project scheduling. *International Journal of Production Research*, to appear.

Wang, J. (2005). Constraint-based schedule repair for product development projects with time-limited constraints. *International Journal of Production Economics*, 95, 399-414.

Zhu, G., Bard, J. and Yu, G. (2005). Disruption management for resource-constrained project scheduling. *Journal of the Operational Research Society*, 56, 365-381.